Slide 1

Administrivia

- Open lab hours announced via e-mail. Updated office hours posted on my Web page, etc.
- Homework 1 on the Web. Due next Friday. One programming problem, so start early.

Slide 2

Minute Essay From Last Lecture(s)

- 8/28 minute essay — a diverse bunch of responses, including:
  - assembly language (MIPS, RISC) in CSCI 1321; also J
  - programming under Windows 98
  - BASIC, QBASIC
  - programming using textpad, notepad, edlin, vi
  - TI calculator
- 8/30 minute essay — see last slide of online notes.

Slide 3

Overview of Hardware, Continued

- Simplified view of hardware (as it appears to programmers) — processor(s), memory, I/O devices, bus.
- Next few sections talk about each component — what it does (from user's point of view) and low-level interface to software.

Slide 4

Processors — Recap

- Basics — instruction set (primitive operations), basic fetch/decode/execute cycle, registers.
- Additional features that help in writing effective operating systems — interrupt mechanism, dual-mode operation and privileged instructions, memory protection, timer.

Slide 5

Memory Hierarchy

- In a perfect world — fast, big, cheap, as permanent as desired.
- In this world — hierarchy of types, from fast but expensive to slow but cheap: registers, cache, RAM, magnetic disk, magnetic tape. (See picture, p. 24.)
- Note also — some types volatile, some non-volatile.

Slide 6

Program Relocation

- At the machine-instruction level, references to memory are in terms of an absolute number. Compilers/assemblers can generate these only by making assumption about where program will reside in memory.
- In the very early days, programs started at 0, so no problem. Now they hardly ever do, so we need a way to relocate programs — when loaded, or "on the fly".
- "On the fly" relocation uses MMU (memory management unit) — which can provide both program relocation and memory protection.

  Logically between CPU and memory, physically usually part of CPU.

  A simple scheme — base and limit registers (described in text). When do values in them need to change?

Slide 7

I/O Devices

- What they provide (from the user's perspective):
  - Non-volatile storage (disks, tapes).
  - Connections to outside world (keyboards, microphones, screens, etc., etc.).
- Distance between hardware and "virtual machine" is large here, so usually think in terms of:
  - Layers of s/w abstraction (as with other parts of o/s).
  - Layers of h/w abstraction too: most devices attached via controller, which provides a h/w layer of abstraction (e.g., "IDE controller").

Slide 8

I/O Basics

- CPU communicates with device controller by reading/writing device registers; device controller communicates with device.
- Memory-mapped I/O versus I/O instructions.
- Polling versus interrupts.
- Functionality for a particular device packaged as "device driver".
- I/O in application programs — make system call.
- Recap: application program $\leftrightarrow$ system call (to o/s) $\leftrightarrow$ device driver $\leftrightarrow$ device controller $\leftrightarrow$ device

Slide 9

Operating System Services

- Process management.
- Memory management.
- I/O subsystem.
- File systems.
- Security.
- Shell (discussed last time).

Slide 10

Process Management

- "Process" abstraction to represent one of a collection of "things happening at the same time".

  A working definition — "program in execution" (program code plus associated variables, sequence of states tracking progress through code and changes in variables).
- "Concurrent" execution via interleaving of actions.

  In effect, each process has a "virtual CPU", with the actual CPU repeatedly suspending one process to work on another ("context switch").
- O/s must provide a way to manage this, including ways to create processes, allow/force them to terminate, communicate among them (e.g., to coordinate/synchronize).

Slide 11

Memory Management

- Managing physical memory:
  - How to divide it up among processes/programs/users — each has an "address space" of memory it can access.
  - How to protect each process's memory from other processes (requires h/w support, but managed by o/s).
- Managing address spaces (virtual memory):
  - Originally, address space limited by size of physical memory.
  - "Virtual memory" allows bigger address spaces, by shuffling data between disk and physical memory.

Slide 12

I/O Subsystem

- Encapsulates messy low-level details.
- Allows sharing of I/O devices among programs/users.

Slide 13

File Systems

- "File system" abstraction, including:
  - "File" abstraction — collection of related information, possibly with associated ownership, permissions, timestamps, etc.
  - "Directory" abstraction.
  - "Path names" — absolute and relative.
  - "Opening a file" — connecting program to file (check permissions, etc., return "file descriptor").
- Additional Unix ideas/terms:
  - Mounting filesystems.
  - Special files — idea is to treat other devices (e.g., printers) like files.
  - Pipes — connections between processes that can be treated like file.

Slide 14

Security

- Protect users/applications from each other.
- Protect users/applications from the outside world.

Slide 15

Minute Essay

- We've talked a lot about what benefits an operating system provides. Can you think of situations, though, in which you wouldn't want one?

Slide 16

Minute Essay Answer

- One possibility: If you only ever want to run one application, it might be simpler and more efficient to just build into it whatever parts of normal o/s functionality are needed.